Fair share is not enough: measuring fairness in scheduling with cooperative game theory

Piotr Skowron and Krzysztof Rzadca

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland,
p.skowron@mimuw.edu.pl, krzadca@mimuw.edu.pl

Abstract. We consider the problem of fair scheduling in a multi-organizational system in which organizations contribute their own resources to the global pool and the jobs to be processed on the common resources. We consider on-line, non-clairvoyant scheduling of sequential jobs without preemption. To ensure that the organizations are willing to cooperate the scheduling algorithm must be fair.
To characterize fairness, we use a cooperative game theory approach. The contribution of an organization is computed based on how this organization influences the utility (which can be any metric, e.g., flow time, turnaround, resource allocation) of all organizations. Formally, the contribution of the organization is its Shapley value in the cooperative game. The scheduling algorithm should ensure that the contributions of the organizations are close to their utilities. Our previous work proves that this problem is NP-hard and hard to approximate.
In this paper we propose a heuristic scheduling algorithm for the fair scheduling problem. We experimentally evaluate the heuristic and compare its fairness to fair share, round robin and the exact exponential algorithm. Our results show that fairness of the heuristic algorithm is close to the optimal. The difference between our heuristic and the fair share algorithm is more visible on longer traces with more organizations. These results show that assigning static target shares (as in the fair share algorithm) is not fair in multi-organizational systems and that instead dynamic measures of organizations' contributions should be used.

keywords: fair scheduling, game theory, algorithm

1 Introduction

A large fraction of contemporary supercomputing resources is run by consortia of independent organizations: from local supercomputing centers shared by a few research groups to international grids, such as Grid5000, DAS or Planet-lab. Each participating organization grants access to its resources to other members of the consortium; in return, an organization expects to be given a fair access to other resources. The role of the consortium is to coordinate access to the resources through a scheduler. Fairness is crucial to the existence of such systems: if an organization feels that it is treated unfairly, it may quit the consortium, thus reducing the pool of the resources accessible by others.

Fairness is one of the key problems in scheduling. Most existing approaches [2,6–10,18] are based on distributive fairness: agents (users, projects or organizations) are assigned a target share of the available resources. The scheduler's goal is to produce

schedules with an average utilization per agent close to the target share. However, distributive fairness does not correspond with agents' goals: an agent, rather than being given an equal share of resources, wants its jobs to be completed fast. Alternative approaches consider agents' utilities (and, sometimes, possible actions of agents). [11] proposes an axiomatic characterization of fairness based on multi-objective optimization; [16] applies this concept to scheduling in a multi-organizational system. [3,4] optimizes the global makespan with an additional constraint that each organization must be at least as well-off as if it acted alone. See [17] for more detailed related work.

In our previous work [17], we considered the problem of fairness by determining the Shapley value of each organization (we summarize these results in Section 3). The Shapley value is a concept commonly used in the cooperative game theory. For an organization, its Shapley value expresses the relative value of the organization to the others. Thus, it represents a fair amount of utility an organization should get from the schedule. In contrast to other works [1,12–14] using monetary valuations for jobs, we proposed to compute the Shapley value directly as a function of how organization's processors increase other organization utilities; and how organization's jobs decrease others utilities. The problem is that an exact scheduling algorithm that produces schedules approximating the Shapley value is exponential ($O(3^n)$).

The contribution of this paper is the following. First, we propose a practical heuristic that schedules jobs according to an estimated Shapley value (Algorithm DIRECTCONTR in Section 4). The heuristic estimates the contribution of an organization by the number of CPU-timeunits an organization contributes for computing jobs of other organizations; the algorithm schedules the jobs to minimize the maximal difference between the utility and the contribution over all organizations. Second, we conduct extensive simulation experiments to verify fairness of commonly-used scheduling algorithms (Section 5). The experiments show that although the fair share algorithm is considerably better than round robin (which does not aim to optimize fairness), our heuristic constantly outperforms fair share, being close to the optimal algorithm and the randomized approximation algorithm. The main conclusion is that ensuring that each party is given a fair share of resources (the distributive fairness approach) might not be sufficient in systems with dynamic job arrival patterns. An algorithm based on the Shapley value, that explicitly considers the organization's impact on other organizations' utilities, produces more fair schedules.

2 The Scheduling Model

We consider a multi-organizational model in which the organizations *may* cooperate; the set of cooperating organizations is called a *coalition* and denoted as $\mathcal{C}$. Every subset of a coalition is also a coalition; however, to emphasize that we are considering a subset of the organizations from a particular $\mathcal{C}$, we will refer to such subsets as to *subcoalitions* of $\mathcal{C}$. Each organization $O^{(u)}$ participating in a coalition $\mathcal{C}$ is contributing its local resources (its processors) to the coalition's global pool. In return, each organization from $\mathcal{C}$ can use all the processors from the coalition's pool to process its own jobs.

The jobs of the organizations may compete for the processors (this happens when in a given time moment there are more jobs waiting for execution than the total number

of free processors), so the organizations need to agree on the scheduling algorithm. Each organization wants its jobs to be processed as fast as possible. The satisfaction of an organization from a schedule can be quantified by a utility function. The utility function of the organization can be any metric that depends on the completion times of the jobs owned by this organization. The classic utilities in the scheduling theory are: flow time, tardiness, turnaround, resource utilization, etc. Hereinafter we will use ψ when referring to the utility function.

The total utility of the organizations participating in coalition $\mathcal{C}$ is called the *value of the coalition* and denoted as $v(\mathcal{C})$. Thus, $v(\mathcal{C}) = \sum_{O^{(u)} \in \mathcal{C}} \psi(O^{(u)})$.

We consider on-line, non-clairvoyant scheduling of sequential jobs. The started jobs cannot be stopped, canceled, or preempted. The organizations decide about the order of processing their own jobs: the jobs of a single organization must be executed in the order they were presented by the organization. The processors are identical.

3 Fairness by the Shapley Value

In this section we describe our approach to fair scheduling by computing the Shapley value. The section summarizes the theoretical results from our previous work [17].

3.1 Computing the Shapley Value

The core idea of our approach lies in computing the effective influence an organization has on the utility of other organizations. The standard, budget-based approaches, when computing the load of an organization, just compute the number of CPU-seconds consumed by the jobs belonging to the organization. This approach ignores the fact that the resources used in peak load periods should be comparatively more expensive than the resources used during low load periods. Similarly, resources contributed by an organization are more valuable during peak load times than when other resources are already idle. Their value directly stems from their ability to execute waiting jobs and thus improve the overall performance.

Taking this approach, the *marginal contribution* of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ ($O^{(u)} \notin \mathcal{C}$) is $v(\mathcal{C} \cup \{O^{(u)}\}) - v(\mathcal{C}))$, i.e., the difference of the total utility of organizations belonging to $\mathcal{C}$ (including $O^{(u)}$) when $O^{(u)}$ joins $\mathcal{C}$. Intuitively, the marginal contribution of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ measures how the presence of the organization $O^{(u)}$ influences (increases or decreases) the completion times of the jobs (the utility) of all organizations participating in $\mathcal{C}$.

The *contribution* $\phi^{(u)}(\mathcal{C})$ of the organization $O^{(u)}$ is its Shapley value. Intuitively, the Shapley value expresses the relative worth of an organization. Formally, let $\mathcal{L}_\mathcal{C}$ denote all orderings of the organizations from the coalition $\mathcal{C}$. Each ordering $\prec_\mathcal{C}$ can be associated with a permutation of the set $\mathcal{C}$, thus $|\mathcal{L}_\mathcal{C}| = |\mathcal{C}|!$. For the ordering $\prec_\mathcal{C} \in \mathcal{L}_C$ we define $\prec_\mathcal{C} (O^{(i)}) = \{O^{(j)} \in \mathcal{C} : O^{(j)} \prec_\mathcal{C} O^{(i)}\}$ as the set of all organizations from $\mathcal{C}$ that precede $O^{(i)}$ in the order $\prec_\mathcal{C}$. The Shapley value can be expressed [15] in the following form:

$$\phi^{(u)}(v(\mathcal{C})) = \frac{1}{|\mathcal{C}|!} \sum_{\prec_\mathcal{C} \in \mathcal{L}_\mathcal{C}} \left(v(\prec_\mathcal{C} (O^{(u)}) \cup \{O^{(u)}\}) - v(\prec_\mathcal{C} (O^{(u)}) \right). \qquad (1)$$

When computing the contribution $\phi^{(u)}(\mathcal{C})$ of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ we consider the process of formation of $\mathcal{C}$—we consider that the organizations may join $\mathcal{C}$ in different orders. For each such an order $\prec$, the organization $O^{(u)}$ joins some already formed subcoalition $\mathcal{C}' \subset \mathcal{C}$. This subcoalition $\mathcal{C}'$ consists of the organizations that joined before $O^{(u)}$, and so of the organizations that are before $O^{(u)}$ in $\prec$. The organization $O^{(u)}$ joining $\mathcal{C}' \subset \mathcal{C}$ changes the value of the coalition by $v(\mathcal{C}' \cup \{O^{(u)}\}) - v(\mathcal{C}'))$ (this is the marginal contribution of $O^{(u)}$ to $\mathcal{C}'$). The contribution of the organization $O^{(u)}$ to a coalition $\mathcal{C}$ is the expected marginal contribution of $O^{(u)}$ when the expectation is taken over all orders of the organizations from $\mathcal{C}$.

An ideally-fair scheduling algorithm should ensure that for each organization $O^{(u)}$, its utility is equal to its contribution, $\forall_u \ \psi(O^{(u)}) = \phi^{(u)}(v(\mathcal{C}))$; however, as the scheduling problem is discrete, such a solution may not exist. Instead, the goal should be to construct in each time moment a schedule that is as fair as possible. More formally, an on-line scheduling algorithm, when there is a free processor, should choose a job of an organization $O^{(u)*}$ that, after being scheduled, minimizes the distance of contributions to utilities, $|\sum_u \psi(O^{(u)}) - \phi^{(u)}(v(\mathcal{C}))|$.

The problem is that, in order to compute contribution $\phi^{(u)}(v(\mathcal{C}))$, each of $2^{|\mathcal{C}|}$ possible coalitions must be analyzed. The complexity of the resulting scheduling algorithm is $O(|\mathcal{O}|(2^{|\mathcal{O}|}\sum m^{(u)} + 3^{|\mathcal{O}|}))$ [17].

3.2 Strategy-resilient utility functions

When defining fairness we need to compare values of utility functions. In distributive fairness the utilities of the organizations should be proportional to their weights. In Shapley fairness (Section 3.1) the utilities should be close to the contributions. However, most of the classic utility functions create incentives for organizations to manipulate their workload. An organization can change its utility by e.g. merging, splitting or delaying their jobs. E.g., consider a job released and started in time 0 and completed in time 2. The flow time of the job is 2. If the job is spit into two smaller jobs – one started in time 0 and completed in time 1 and the other started in time 1 and completed in time 2, then the total flow time of the two jobs is 3. Thus, by splitting a job, the organization can later require better service by claiming that it obtained worse service from what it actually got.

A strategy-resilient (non-manipulable) utility function exists [17]. Let σ denote a schedule of the jobs of a given organization. We assume that σ is a set of pairs (s, p), each pair representing a single job; s denotes the start time and p denotes the processing time of a job. If the job is not yet completed (p is not known), we set $p = (t - s)$. A strategy-resilient utility function in time t has the following form:

$$\psi_{sp}(\sigma, t) = \sum_{(s,p)\in\sigma:s\leq t} p\left(t - \frac{s + (s + p - 1)}{2}\right). \tag{2}$$

Intuitively, in ψ_{sp} the jobs are considered as sets of unit-size tasks. Each task obtains a utility proportional to the time in which it completes. If a job completes at time t_c, at time t it gets a utility equal to $(t - t_c)$. This function can be thought of as the throughput of the jobs of an organization. If we consider a fixed set of jobs with equal processing

times, maximization of ψ_{sp} is equivalent to minimization of the flow time [17]. The utility function ψ_{sp} takes into account only the completed jobs and the completed unit-size parts of the jobs (by setting $p = (t - s)$ whenever job is still being processed), thus it is adequate for non-clairvoyant models).

4 Algorithms

In this section we describe the algorithms that we evaluate. We start by a description of the exact exponential fair algorithm REF [17]; we then describe a randomized algorithm RAND [17] approximating REF; our new heuristic DIRECTCONTR; and the reference algorithms ROUNDROBIN and FAIRSHARE in three versions differing by what the algorithm balance: the shares of assigned processors in FAIRSHARE, the utility functions in UTFAIRSHARE and the number of concurrently executed jobs in CURRFAIRSHARE.

REF. This algorithm is a direct implementation of the definition from Section 3.1. It is based on the concepts of utilities and contributions of the organizations. The contribution of the organization is defined by its Shapley value.

The referral algorithm schedules the jobs to ensure that the contributions of organizations are as close to their utilities as possible. Calculating the Shapley value for the organization is NP-hard and hard to approximate [17]. The core difficulty lies in the fact that to calculate the Shapley value in each time moment one has to know the schedules for each subset—with k organizations there are 2^k such subsets. As the result, the practical usage of the algorithm REF is limited and we can only use this algorithm as a benchmark for evaluating other algorithms.

RAND. Taking into account the computational hardness of the algorithm REF, in our previous work [17] we proposed a randomized algorithm. Instead of remembering the schedules for all 2^k subsets, the algorithm for each organization $O^{(u)}$ selects only N random subsets not containing $O^{(u)}$. The contribution of $O^{(u)}$ is calculated based on the marginal contribution of $O^{(u)}$ only to these N subsets (the idea is similar to Monte-Carlo methods for computing the Shapley value). Such approach guarantees that for sufficiently large N with high probability we can get arbitrarily good approximation bounds for the fairness [17]. Although this algorithm has good theoretical properties, it requires a large N to produce high-quality schedules.

DIRECTCONTR (see Algorithm 1). The algorithm keeps for each organization O its utility $\psi_{sp}[O]$ and its estimated contribution $\phi[O]$. The estimate of the contribution of each organization is assessed directly (without considering any subcoalitions) by the following heuristic. On each scheduling event t we consider the processors in a random order and assign waiting jobs to free processors. The job that is started on processor m increases the contribution $\tilde{\phi}$ of the owner of m by the utility of this job.

In the pseudo code, $\mathrm{finUt}[O]$ denotes the number of the unit-size parts of the jobs of the organization O that are completed before t_{prev}. From Equation 2 we know that the utility in time t of the unit-size parts of the jobs of the organization O that are completed before t_{prev} is greater by $(t - t_{prev})\mathrm{finUt}[O]$ than this utility in time t_{prev} (line 7); the utility of the unit-size parts of the job completed between t_{prev} and t is

Algorithm 1: DIRECTCONTR: a heuristic algorithm for Shapley-fair scheduling.

Notation:
own(M), own(J) — the organization owning the processor M, the job J
$wait(O)$ — the set of released, but not-yet scheduled jobs of the organization O at time t

1 Initialize$(\mathcal{C})$:
2 **foreach** $O^{(u)} \in \mathcal{C}$ **do**
3 finUt$[O^{(u)}] \leftarrow 0$; finCon$[O^{(u)}] \leftarrow 0$;
4 $\phi[O^{(u)}] \leftarrow 0$; $\psi[O^{(u)}] \leftarrow 0$;
5 Schedule(t_{prev}, t): // t_{prev} is the time of the previous event
6 **foreach** $O^{(u)} \in \mathcal{C}$ **do**
7 $\phi[O^{(u)}] \leftarrow \phi[O^{(u)}] + (t - t_{prev})$finCon$[O^{(u)}]$;
8 $\psi[O^{(u)}] \leftarrow \psi[O^{(u)}] + (t - t_{prev})$finUt$[O^{(u)}]$;
9 $\gamma \leftarrow$ generate a random permutation of the set of all processors;
10 **foreach** $m \in \gamma$ **do**
11 **if *not*** FreeMachine(m, t) **then**
12 $J \leftarrow$ RunningJob(m) ;
13 finUt[own(J)] $\leftarrow$ finUt[own(J)] $+ t - t_{prev}$;
14 finCon[own(m)] $\leftarrow$ finCon[own(m)] $+ t - t_{prev}$;
15 ϕ[own(J)] $\leftarrow \phi$[own(J)] $+ \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
16 ψ[own(m)] $\leftarrow \psi$[own(m)] $+ \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$;
17 **foreach** $m \in \gamma$ **do**
18 **if** FreeMachine(m, t) ***and*** $\bigcup_{O^{(u)}} wait(O^{(u)}) \neq \emptyset$ **then**
19 $org \leftarrow \text{argmax}_{O^{(u)}: wait(O^{(u)}) \neq \emptyset}(\phi[O^{(u)}] - \psi[O^{(u)}])$;
20 $J \leftarrow$ first waiting job of org ;
21 startJob(J, m) ;
22 finUt$[org] \leftarrow$ finUt$[org] + 1$;
23 finCon[own(m)] $\leftarrow$ finCon[own(m)] $+ 1$;

equal to $\sum_{i=1}^{t-t_{prev}} i = \frac{1}{2}(t - t_{prev})(t - t_{prev} + 1)$ (line 15). Similarly, $\text{finCon}[O]$ denotes the number of the completed unit-size parts of the jobs processed on the processors of the organization O. The algorithm updates the utilities and the estimates of the contributions. The waiting jobs are assigned to the processors in the order of decreasing differences $(\phi - \psi)$ of the issuing organizations (similarly to REF).

ROUNDROBIN. The algorithm cycles through the list of organizations to determine the job to be started.

FAIRSHARE [10]. This is perhaps the most popular scheduling algorithm using the idea of distributive fairness. Each organization is given a target weight (a *share*). The algorithm tries to ensure that the resources used by different organizations are proportional to their shares. More formally, whenever there is a free processor and some jobs waiting for execution, the algorithm sorts the organizations in the ascending order of the ratios: the total time of the processor already assigned for the jobs of the organization divided by its share. A job from the organization with the lowest ratio is started.

In all versions of fair share, in the experiments we set the target share to the fraction of processors contributed by an organization to the global pool.

UTFAIRSHARE. This algorithm uses the same idea as FAIRSHARE. The only difference is that UTFAIRSHARE tries to balance the utilities of the organizations instead of their resource allocation. Thus, in each step the job of the organization with the smallest ratio of utility to share is selected.

CURRFAIRSHARE. This version of the fair share algorithm does not keep any history; it only ensures that, for each organization, the number of currently executing jobs is proportional to its target share.

5 Simulation experiments

5.1 Settings

To run simulations, we chose the following workloads from the Parallel Workload Archive [5]: 1. LPC-EGEE[1] (cleaned version), 2. PIK-IPLEX[2], 3. RICC[3], 4. SHARCNET-Whale[4]. We selected traces that closely resemble sequential workloads (in the selected traces most of the jobs require a single processor). We replaced parallel jobs that required $q > 1$ processors with q copies of a sequential job having the same duration.

In each workload, each job has a user identifier (in the workloads there are respectively 56, 225, 176 and 154 distinct user identifiers). To distribute the jobs between the organizations we uniformly distributed the user identifiers between the organizations; the job sent by the given user was assigned to the corresponding organization.

Because REF is exponential, the experiments are computationally-intensive; in most of the experiments, we simulate only 5 organizations.

The users usually send their jobs in consecutive blocks. We also considered a scenario when the jobs are uniformly distributed between organizations (corresponding to a case when the number of users within organizations is large, in which case the distribution of the jobs should be close to uniform). These experiments led to the same conclusions, so we present only the results from the case when the user identifiers were distributed between the organizations.

For each workload, the total number of the processors in the system was equal to the number originally used in the workload (that is 70, 2560, 8192 and 3072, respectively). The processors were assigned to organizations so that the counts follow Zipf and (in different runs) uniform distributions.

For each algorithm, we compared the vector of the utilities (the utilities per organization) at the end of the simulated time period (a fixed time t_{end}): $\boldsymbol{\psi}$ with the vector of the utilities in the ideally fair schedule $\boldsymbol{\psi}^*$ (computed by REF). Let p_{tot} denote the total number of the unit-size parts of the jobs completed in the fair schedule returned by REF, $p_{tot} = \sum_{(s,p)\in\sigma^*:s\leq t_{end}} \min(p, t_{end} - s)$. We calculated the difference $\Delta\psi = |\boldsymbol{\psi} - \boldsymbol{\psi}^*| = \sum_{O^{(u)}} (\psi^{(u)} - \psi^{(u),*})$ and compared the values $\Delta\psi/p_{tot}$ for different algorithms. The value $\Delta\psi/p_{tot}$ is the measure of the fairness that has an intuitive

[1] www.cs.huji.ac.il/labs/parallel/workload/l_lpc/index.html
[2] www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/index.html
[3] www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html
[4] www.cs.huji.ac.il/labs/parallel/workload/l_sharcnet/index.html

Table 1. The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^4$.

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
ROUNDROBIN	238	353	6	33	145	38	2839	357
RAND ($N = 15$)	8	21	0.014	0.01	6	6	162	187
DIRECTCONTR	5	11	0.02	0.15	10	7	537	303
FAIRSHARE	16	25	0.3	1.38	13	8	626	309
UTFAIRSHARE	16	25	0.3	1.38	38	67	515	284
CURRFAIRSHARE	87	106	0.3	1.58	145	80	1231	243

Table 2. The average delay (or the speed up) of jobs due to the unfairness of the algorithm $\Delta\psi/p_{tot}$ for different algorithms and different workloads. Each row is an average over 100 instances taken as parts of the original workload. The duration of the experiment is $5 \cdot 10^5$.

	LPC-EGEE		PIK-IPLEX		SHARCNET-Whale		RICC	
	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.	Avg	St. dev.
ROUNDROBIN	4511	6257	242	1420	404	1221	10850	13773
RAND ($N = 15$)	562	1670	1.3	7	26	158	771	1479
DIRECTCONTR	410	1083	0.2	1.4	60	204	1808	3397
FAIRSHARE	575	1404	2.3	12	94	307	2746	4070
UTFAIRSHARE	888	2101	1.2	5	120	344	4963	6080
CURRFAIRSHARE	1082	2091	2.2	11	180	805	5387	9083

interpretation. Since delaying each unit-size part of a job by one time moment decreases the utility of the job owner by one, the value $\Delta\psi/p_{tot}$ gives the average unjustified delay (or unjustified speed-up) of a job due to the unfairness of the algorithm.

5.2 Results

We start with experiments on short sub-traces of the original workloads. We randomly selected the start time of the experiment t_{start} and set the end time to $t_{end} = t_{start} + 5 \cdot 10^4$. For each workload we run 100 experiments (on different periods of workloads of length $5 \cdot 10^4$). The average values of $\Delta\psi/p_{tot}$, and the standard deviations are presented in Table 1.

From this part of the experiments we conclude that: (i) The algorithm RAND is the most fair algorithm regarding the fairness by the Shapley Value; but RAND is the second most computationally intensive algorithm (after REF). (ii) All the other algorithms are about equally computationally efficient. The algorithm DIRECTCONTR is the most fair. (iii) The algorithm FAIRSHARE, which is the algorithm mostly used in real systems, is not much worse than DIRECTCONTR. (iv) Arbitrary scheduling algorithms like ROUNDROBIN may result in unfair schedules. (v) The fairness of the algorithms may depend on the workload. In RICC the differences are much more visible than in PIK-IPLEX. Thus, although DIRECTCONTR and FAIRSHARE are usually comparable, on some workloads the difference is important.

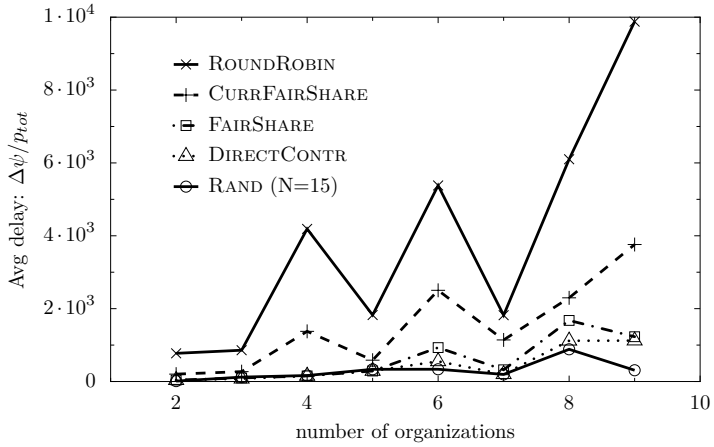

Fig. 1. The effect of the number of the organizations on ratio $\Delta\psi/p_{tot}$.

In the second series of experiments, we verified the effect of the duration of the simulated workload on the resulting fairness measure (the ratio $\Delta\psi/p_{tot}$). As we changed the duration of the experiments from $5 \cdot 10^4$ to $5 \cdot 10^5$, we observed that the unfairness ratio $\Delta\psi/p_{tot}$ was increasing. The value of the ratio for $t_{end} - t_{start} = 5 \cdot 10^5$ are presented in Table 2. The relative quality of the algorithms is the same as in the previous case. Thus, all our previous conclusions hold. However, now all the algorithms are significantly less fair than the exact algorithm. Thus, in long-running systems the difference between the approaches becomes more important. If there are a few organizations, the exact REF or the randomized RAND algorithms should be used. In larger systems, when the computational cost of these is too high, DIRECTCONTR clearly outperforms FAIRSHARE.

Last, we verified the effect of the number of the organizations on the ratio $\Delta\psi/p_{tot}$. The results from the experiments conducted on LPC-EGEE data set are presented in Figure 1. As the number of organizations increases, the unfairness ratio $\Delta\psi/p_{tot}$ grows; thus the difference between the algorithms is more significant. This confirms our previous conclusions. The relative fairness of the algorithms is the same as in our previous experiments.

6 Conclusions

In this paper we present DIRECTCONTR, a heuristic algorithm for the problem of Shapley-fair scheduling. We conduct extensive experimental evaluation of the fairness of our algorithm comparing it to other algorithms used in real systems. We conclude that the randomized algorithm is the closest to the referral exponential algorithm, yet it is also the most computationally intensive. Among computationally-tractable algorithms, DIRECTCONTR, our heuristic, is the closest to the referral algorithm, although on shorter workloads with relatively few organizations, the fair share algorithm is similar. The difference between algorithms becomes significant in longer-running systems with

many organizations. The main conclusion from our work is that in multi-organizational systems, the distributive fairness used by fair share does not result in truly-fair schedules; our heuristic better approximates the Shapley-fair schedules.

References

1. T. E. Carroll and D. Grosu. Divisible load scheduling: An approach using coalitional games. In *ISPDC, Proceedings*, 2007.
2. H. M. Chaskar and U. Madhow. Fair scheduling with tunable latency: a round-robin approach. *IEEE/ACM Trans. Netw.*, 11(4):592–601, 2003.
3. J. Cohen, D. Cordeiro, D. Trystram, and F. Wagner. Multi-organization scheduling approximation algorithms. *Concurrency and Computation: Practice and Experience*, 23(17):2220–2234, 2011.
4. P.-F. Dutot, F. Pascual, K. Rzadca, and D. Trystram. Approximation algorithms for the multi-organization scheduling problem. *IEEE Transactions on Parallel and Distributed Systems*, 22:1888 – 1895, 2011.
5. D. G. Feitelson. Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/.
6. P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *SIGCOMM, Proceedings*, pages 157–168, 1996.
7. A. Gulati and I. Ahmad. Towards distributed storage resource management using flow control. *SIGOPS Oper. Syst. Rev.*, 42(6):10–16, 2008.
8. A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *FAST, Proceedings*, Feb. 2009.
9. W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *SIGMETRICS Perform. Eval. Rev.*, 32(1):37–48, 2004.
10. J. Kay and P. Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.
11. M. M. Kostreva, W. Ogryczak, and A. Wierzbicki. Equitable aggregations and multiple criteria analysis. *European Journal of Operational Research*, 158(2):362–377, 2004.
12. L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. In *PCCC, Proceedings*, pages 1–8, 2011.
13. D. Mishra and B. Rangarajan. Cost sharing in a job scheduling problem using the shapley value. In *EC, Proceedings*, pages 232–239, 2005.
14. H. Moulin. On scheduling fees to prevent merging, splitting, and transferring of jobs. *Math. Oper. Res.*, 32(2):266–283, May 2007.
15. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, 1994.
16. K. Rzadca, D. Trystram, and A. Wierzbicki. Fair game-theoretic resource management in dedicated grids. In *CCGRID, Proceedings*, 2007.
17. P. Skowron and K. Rzadca. Non-monetary fair scheduling — cooperative game theory approach. In *SPAA (see also the extended arxiv version)*, 2013.
18. Y. Wang and A. Merchant. Proportional-share scheduling for distributed storage systems. In *FAST, Proceedings*, pages 4–4, 2007.

Acknowledgements. This work is partly supported by Polish National Science Center Sonata grant UMO-2012/07/D/ST6/02440